Flexible Integration of Planning and Information Gathering 

David Camacho, Daniel Borrajo, José M. Molina, and Ricardo Aler

Universidad Carlos III de Madrid, Computer Science Department, Avenida de la Universidad nº 30, CP 28911, Leganés, Madrid, Spain
{dcamacho, dborrajo, molina}@ia.uc3m.es, aler@inf.uc3m.es

Abstract. The evolution of the electronic sources connected through wide area networks like Internet has encouraged the development of new information gathering techniques that go beyond traditional information retrieval and WEB search methods. They use advanced techniques, like planning or constraint programming, to integrate and reason about hetereogeneous information sources. In this paper we describe MAPWEB, a multiagent framework that integrates planning agents and WEB information retrieval agents. The goal of this framework is to deal with problems that require planning with information to be gathered from the WEB. MAPWEB decouples planning from information gathering, by splitting a planning problem into two parts: solving an abstract problem and validating and completing the abstract solutions by means of information gathering. This decoupling allows also to address an important aspect of information gathering: the WEB is a dynamic medium and more and more companies make their information available in the WEB everyday. The MAPWEB framework can be adapted quickly to these changes by just modifying an abstract planning domain and adding the required information gathering agents. For instance, in a travel assistant domain, if taxi companies begin to offer WEB information, it would only be necessary to add new planning operators related to traveling by taxi, for a more complete travel domain. This paper describes the MAPWEB planning process, focusing on the aforementioned flexibility aspect.

1 Introduction

In recent years there has been a lot of work in Web information gathering [1, 5–8]. Information gathering intends to integrate a set of different information sources with the aim of querying them as if they were a single information source [6]. Many different kinds of systems, named *mediators*, have been developed. They try to integrate information from multiple distributed and heterogeneous information sources, like database systems, knowledge bases, web servers, electronic repositories... (an example is the *SIMS* [7] architecture). In order that these systems are practical, they must be able to optimize the query process by selecting the most appropriate WEB sources and ordering the queries. For this purpose, different algorithms and paradigms have been developed. For instance, *Planning by Rewriting* (*PbR*) [1] builds queries by using planning techniques.

Other examples of information gathering systems are Ariadne [7], Heracles [8], WebPlan [5].

Some of the previous approaches use planning techniques to select the appropriate WEB sources and order the queries to answer generic user queries. That is, they use planning as a tool for selecting and sequencing the queries. In this paper we describe MAPWEB, an information gathering system that also uses planning, but with a different purpose (some preliminary work can be found in [2, 3]). MAPWEB uses planning for both determining the appropriate generic sources to query and solving actual planning problems. For instance, in this paper, the MAPWEB framework is applied to a travel planning assistant domain (e-tourism),[1] where the user needs to find a plan to travel among several places. Each plan not only determines what steps the user must perform, but which information sources should be accessed. For instance, if a step is to go from A to B by plane, the system provides the user the information of what airplane companies should be consulted for further information. This domain is similar to the travel planning assistant built using the Heracles framework. However, Heracles constrained network, which is a kind of plan schema, needs to be reprogrammed everytime the planning domain changes. MAPWEB tries to be more flexible by using planning techniques to create the plans. For instance, if it is desired to add a new information source to the system, it is only necessary to change the planning domain instead of reprogramming the plan schema by hand. For instance, if taxi fares were made suddenly available in the WEB, it would only be necessary to add a move-by-taxi operator along with the associated WebAgent.[2] Actually, MAPWEB can handle planning operators which are not associated to any information source (because, for instance, the information on a given topic is not yet available). In that case, plans will contain steps with no detailed information. This is useful, because even if no specific information is supplied, at least the user is told that he can fulfill that step by any means.

The paper is structured as follows. Section 2 describes MAPWEB architecture. Section 3 explains in detail the abstract planning process. Section 4 evaluates empirically the system. Finally, Section 5 summarizes the conclusions and future lines of work.

2 MAPWEB System Architecture

MAPWEB is structured into several logic layers whose purpose is to isolate the user from the details of problem solving and WEB access. More specifically, we considered four layers between users and the WEB: *the physical world* (the users), *the reasoning layer* (that includes user agents, planning agents, and control agents), *the access information layer* (that contains WebAgents to retrieve the desired information), and *the information world* (which represents the available information). This four-layer architecture can be seen in Figure 1.

[1] This domain is a modified version of the Logistics domain.

[2] A WebAgent is an information agent specialized in consulting a particular information source.

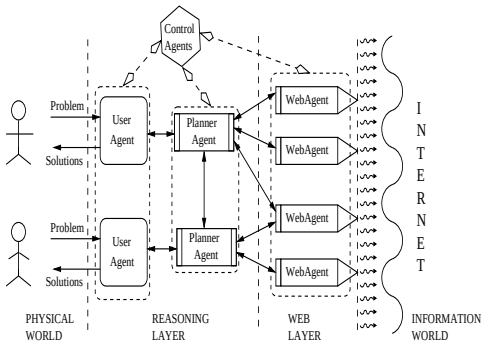

Fig. 1. MAPWeb three-layer architecture.

MAPWeb deploys this architecture using a set of heterogeneous agents. Next, each of these types of agents will be described:

- **UserAgents**: They pay attention to user queries and display to users the solution(s) found by the system. When an UserAgent receives problem queries from the users, it sends them to the PlannerAgents and when they answer back with the plans, the UserAgent provides the solutions to the user.
- **ControlAgents**: They handle several control functions like the insertion and deletion of agents in the system and communication management.
- **PlannerAgents**: They receive an user query, build an abstract representation of it, and solve it by means of planning. Then, the PlannerAgents fill in the information details by querying the WebAgents. The planner that has been used by the PlannerAgents is PRODIGY4.0 [9].
- **WebAgents**: Their main goal is to fill in the details of the abstract plans obtained by the PlannerAgents. They obtain that information from the Web.

The way these agents cooperate is as follows. First, the user interacts with the UserAgent to input his/her query. The query captures information like the departure and return dates and cities, one way or return trip, maximum number of transfers, and some preference criteria. This information is sent to the PlannerAgent, which transforms it into a planning problem. This planning problem retains only those parts that are essential for the planning process, which is named the *abstract representation* of the user query. PRODIGY4.0 generates several abstract solutions for the user query. The planning operators in the abstract solutions require to be completed and validated with actual information which is retrieved from the Web. To accomplish this, the PlannerAgent sends information queries to specialized WebAgents, that return several records for every information query. Then, the PlannerAgent integrates and validates the solutions and returns the data to the UserAgent, which in turn displays it to the user. MAPWeb agents use a subset of the KQML speech acts [4]. The whole process will be described in full detail in the next section.

3 The Planning Process

As mentioned before, in MAPWeb, the information gathering process is carried out by a set of WebAgents, but this process is guided by the PlannerAgent that reasons about the requested problem and the different information sources that are available.

The planning process is divided into two parts: solving an abstract problem, and completing it with information gathered from the Web. Planning is decoupled this way because of two reasons:

- The abstract planning problem is easier to solve by classical planners. This is because if all the information about all the available flights, all possible trains, etc. was included in the planning process, planning would be unfeasible.
- It is not necessary to access the Web during the planning process. Queries to the WebAgents are carried out only when abstract plans are ready. This allows to reduce the number of queries, because only those queries that are required by the solution are ever made.

Planning works as follows. First, the PlannerAgent receives a query from UserAgent. This query is analyzed and translated into an abstract planning problem. Second, the PlannerAgent uses its own skills and knowledge about the problem and tries to solve it. If the solving process is successful, the PlannerAgent generates a set of abstract solutions. These solutions are too general and only have the essential information for the planning process, so they need specific information to be completed and validated. The PlannerAgent builds a set of information queries (queries to other agents in the system to request specific information). It is important to try to optimize the number of queries due to the high number of possible instantiations. When the queries have been built, the PlannerAgent selects the set of WebAgents that will be asked. Finally, when the WebAgents answer with the information found in the Web (if WebAgents are successful) the PlannerAgent integrates all the specific information with the abstract solutions to generate the final solutions that will be sent to the UserAgent. In Figure 2 the modular description of the planning process is shown.

The next subsections explain this process in detail by focusing in the data structures used by each of the relevant agents: the user query generated by the UserAgent, the abstract problem, the abstract solutions, the specific knowledge used by the PlannerAgent, and finally the specific information records retrieved by the WebAgents.

3.1 The User Query

The planning process starts when the user supplies a problem to be solved. A user query is a sequence of stages. Each stage is a template that represents a leg of the trip, and contains several fields to be filled by the user. Table 1 shows an instance of a possible user query. It will be used to illustrate the rest of the

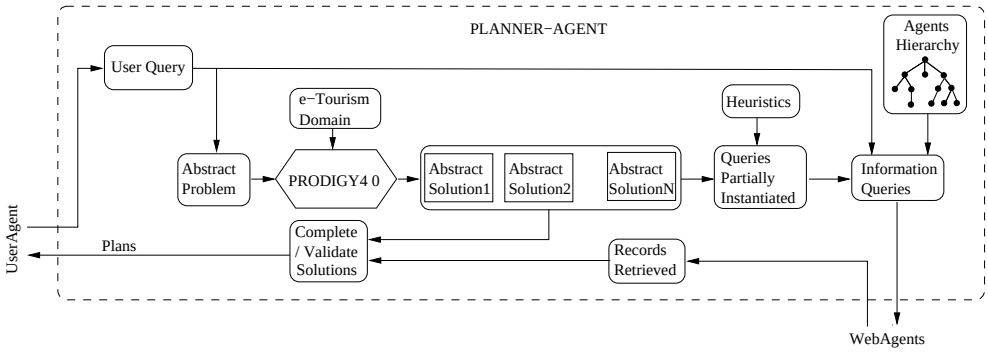

Fig. 2. Planning Process developed by the PlannerAgent. The user query is transformed into an abstract planning problem, which is subsequently solved by PRODIGY4.0. Each solution is partially instantiated by means of domain dependent heuristics. Every operator in a solution generates several WEB queries, which are sent to the appropriate WebAgents by using the agent hierarchy. The agents return several records, that are used to complete and validate the abstract solutions.

article. This query is then sent to the PlannerAgent. Besides the information shown in Table 1, the user can specify the locations inside the city where s/he wants to start or end the trip (like an airport, a train station, or a bus station). This is done by means of the user interface provided by the UserAgent.

Table 1. A user problem to go from Turin to Toledo by airplane or train.

Leg	Stage	Date	Restrictions	N° Transfers
1	Turin → Madrid	Sep. 11th	Plane or train	0 or 1
2	3 nights stay	Sep. 11th	< 15.000 pts	-
3	Madrid → Toledo	Sep. 14th	Plane or train	0 or 1
4	Toledo → Turin	Sep. 14th	Plane or train	0 or 1

3.2 The planning domain and the abstract solutions

The PlannerAgent transforms the user query into an abstract problem. This is done as follows. First, it defines an abstract city. This city includes all possible local transports, but only the long range transport terminals that the user wishes to use are included. Then, this abstract city is copied as many times as the maximum number of transfers supplied by the user. It is important to remark that the cities are abstract cities (i.e. they have no attached names, so they are present in the abstract plan to represent the initial, intermediate, and final travel points). The rest of details provided by the user are ignored at this stage. The abstract problem represents the initial state and the goals of the problem that are the inputs to PRODIGY4.0.

In order to solve abstract problems, PRODIGY4.0 requires a domain where the planning operators are described. Using planning at this stage (instead of using pre-programmed plans) provides two main advantages:

1. *Flexibility:* the system can be adapted to many different versions of travel domains and problems by just changing the domain description or the abstract problem generation method, respectively.
2. *Easy integration of new* WEB *sources.* The WEB is a dynamic medium: more and more companies make their information available in the WEB everyday. If a new information source (like taxi fares) is made available, MAPWEB can be adapted quickly by just adding a new planning operator and establishing a relation with a WebAgent specialized in gathering the information from the WEB.

The abstract problem obtained from Table 1 would be given to the PlannerAgent planner (PRODIGY4.0) which would obtain several possible abstract solutions. In this case, the planner would reply with the plans shown in Figures 3 (solutions with 0 transfers) and 4 (1 transfer solutions).

```
Problem: 0-Transfers
Solution 1:
 <travel-by-airplane user1 plane0 airport0 airport2>
 <move-by-local-transport user1 lbus2 bustop20 trainstat21 city2>

Solution 2:
 <move-by-local-transport user1 lbus0 bustop00 trainstat01 city0>
 <travel-by-train user1 train0 trainstat0 trainstat2>
```

Fig. 3. Abstract solutions generated by PRODIGY4.0 for Leg 1 with 0-Transfer.

```
Problem: 1-Transfers
Solution 1:
 <travel-by-airplane user1 plane0 airport0 airport1>
 <move-by-local-transport lbus1 bustop10 trainstat11 city1>
 <travel-by-train user1 train1 trainstat1 trainstat2>

Solution 2:
 <travel-by-airplane user1 plane0 airport0 airport1>
 <travel-by-airplane user1 plane1 airport1 airport2>
 <move-by-local-transport lbus2 bustop20 trainstat20 city2>
```

Fig. 4. Abstract solutions generated by PRODIGY4.0 for Leg 1 with 1-Transfers.

This is a set of abstract plans that contain no actual details. Some of the plan steps might not even be possible because, for instance, there are no companies linking two cities. Therefore, those plans need to be *validated* and *completed.* The PlannerAgent acomplishes this task in the following way:

1. The abstract steps in the solution contain unbound variables that relate to transfer cities. They need to be bound before the WebAgents are queried.

The PlannerAgent restricts the number of bindings by applying a *geographic heuristic*. This is achieved as follows:

- If the origin and arrival cities belong to the same country, only the cities in that country are considered as possible transfer cities.
- Else, if the origin and arrival cities belong to the same continent, only the cities of that continent are considered.
- Otherwise, all cities are considered.

In the case of the first leg of the trip, as Turin and Madrid belong to Europe, we extract the cities that belong to this continent (currently, about 30). Table 2 displays the queries that would be generated in this case.

Table 2. Queries partially instantiated.

Query send to the WebAgents	N° Transfers
(travel-by-airplane user1 plane0? Turin Toledo)	0
(travel-by-train user1 train0? Turin Toledo)	0
(travel-by-airplane user1 plane0? Madrid Turin)	0
(travel-by-train user1 train0? Madrid Toledo)	0
(travel-by-airplane user1 plane0? Turin Alicante)	1
(travel-by-airplane user1 plane0? Turin Barcelona)	1
(travel-by-airplane user1 plane0? Turin Paris)	1
(travel-by-train user1 train0? Turin Madrid)	1
...	...

2. Planning operators of the abstract solutions and WEB sources are related by means of a WebAgent *hierarchy*. This *hierarchy* is used by the PlannerAgent to select the relevant WebAgents that will be used to obtain the information. This hierarchy allows the PlannerAgent to know which WebAgents know how to retrieve the required information. In Figure 5 a description of this hierarchy is shown.

```
Agent
    |___UserAgent0
    |---ReasonerAgent
                    |___PlannerAgent0
    |___WebAgents
                |___Travel
                            |___Fly: WebAgent-Iberia,WebAgent-Amadeus-Flight
                            |___Train: WebAgent-Renfe,WebAgent-RailEurope
                            |___Bus
                |___Hotel: WebAgent-Amadeus-Hotel
                |___Car
    |___ControlAgents
                    |___ManagerAgent0
                    |___CoachAgent0
```

Fig. 5. Agents Hierarchy. It describes all the available agents in MAPWEB and their information gathering skills.

3. Finally the PlannerAgent uses the previous information to build a set of queries that will sent to the selected WebAgents. If a planning operator is repeated in different abstract solutions, it is only considered once, to avoid repeating queries. For instance, in the solutions for 1-Transfer problems the operator (<`travel-by-airplane user1 plane0 airport0 airport1`>) would be translated as shown in Table 3:

Table 3. Queries partially instantiated to the appropriate WebAgents.

Query send to the WebAgents	WebAgent
(travel-by-airplane user1 plane0? Turin Toledo)	Iberia, Amadeus-Flights
(travel-by-train user1 train0? Turin Toledo)	Renfe, RailEurope
(travel-by-airplane user1 plane0? Turin Alicante)	Iberia, Amadeus-Flights
(travel-by-airplane user1 plane0? Turin Barcelona)	Iberia, Amadeus-Flights
(travel-by-airplane user1 plane0? Turin Paris)	Iberia, Amadeus-Flights
(travel-by-train user1 train0? Turin Madrid)	Renfe, RailEurope
...	...

Those queries (and all the additional information given by the UserAgent) are sent to several WebAgents that know about airplane travel, so that variable `plane0?` is instantiated as well.

3.3 Filling the Abstract Solutions

The information queries are sent to the selected WebAgents with the specific data (departure and arrival times, travel cost, etc...) and the query that the PlannerAgent needs. With this information the WebAgents automatically build the specific WEB query that will be sent to the WEB information sources the agent is specialized in. For every query, each WebAgent will return to the PlannerAgent a list of *records* by filling a template whose structure is shared by all the agents (there are different templates depending on the kind of information required). In Table 4 some of the retrieved flight-records and train-records provided by different WebAgents are shown for the Leg 1 in the example.

Finally, those records are received by the PlannerAgent that will use them to complete the abstract solutions. If the WebAgents return no records for a step of the abstract solution, that particular solution is rejected. However, it is important to remark that if it is known in advance that there are no WEB sources to complete a particular step (for instance, <`MOVE-BY-LOCAL-TRANSPORT taxi ...`>), then the user is told that s/he has to carry out that step, even though no specific information about that step is attached. The set of completed solutions are finally sent to the UserAgent that requested the information.

4 Experimental Evaluation

The aim of this section is to carry out several experiments with MAPWEB to evaluate its performance. First, the example-trip we have used to illustrate the

Table 4. Retrieved records by the WebAgents.

Inf-FLIGHTS	record1	record2	record3	Inf-TRAINS	record1	record2	record3
WebAgent	**Iberia**	**Amadeus**	**Amadeus**	**WebAgent**	**Renfe**	**Renfe**	**Renfe**
air-company	Iberia	Iberia	Portugalia	train-company	RENFE	RENFE	RENFE
http-address	w3.iberia.es	null	null	http-address	w3.renfe.es	w3.renfe.es	w3.renfe.es
flight-id	IB8797	IB8819	NI711	train-id	07054	07056	07058
ticket-fare	70641	null	null	ticket-fare	780	780	780
currency	ESP	ESP	ESP	currency	ESP	ESP	ESP
flight-duration	3h45min	2h00min	2h10min	departure-city	MAD	MAD	MAD
airp-depart-city	TRN	TRN	TRN	departure-date	11-09-01	11-09-01	11-09-01
departure-date	11-09-01	11-09-01	11-09-01	departure-time	6:30	8:30	10:14
airp-arrival-city	MAD	MAD	MAD	arrival-city	TOL	TOL	TOL
return-date	null	null	null	arrival-date	11-09-01	11-09-01	11-09-01
class	Tourist	null	null	arrival-time	7:53	9:47	11:30
n° passengers	1	1	1	class	Tourist	Tourist	Tourist
round-trip	one-way	one-way	one-way				

previous sections will be tested. Second, a set of problems given by the user will be evaluated to analyze the average behaviour of the system.

Table 5 summarizes the example-trip (Turin to Toledo and back). To solve this problem, a team of nine agents was used. It includes all the agents displayed in the agents-hierarchy of Figure 5. In particular, airplane, train, and hotel WebAgents have been used. The next parameters have been measured:

- Validated abstract solutions/abstract solutions ratio (*val.sols/abs.sols*). This value measures how many abstract solutions provided through the planner were validated by the information provided by the information gathering agents.
- Number of instantiated solutions. It shows all the possible solutions to the user problem. The solutions are computed using the gathered records. The PlannerAgent uses the k validated abstract solutions that contain l_i abstract operators. If there are b_{ij} retrieved records for the j-th operator of the i-th solution, then the number of possible instantiated solutions is:

$$\text{Number of solutions} = \sum_{i=1}^{k} \prod_{j=1}^{l_i} b_{ij}$$

- Number of WEB Queries. This represent all the queries made by the WebAgents to retrieve the specific information.
- Number of gathered records (duplicated records are removed).
- Time. It includes planning time and WEB gathering time. It is elapsed time (i.e. the time spent by the WebAgents acting in parallel is not accumulated).

Everyone of the previous parameters is measured for both 0 and 1 transfers (0-T and 1-T). In this example, there are no solutions for the 0 transfers because it is impossible to complete the fourth leg of the trip (there is no way to go from Toledo to Turin directly). On the other hand, there are thousands of possible combinations when 1 transfer is allowed. It is important to remark that even though when 1 transfer is used, it takes several thousand seconds to find the solutions, only 1 second per leg is spent for actual planning.

Table 5. MAPWEB request for the example-trip, with 0 and 1 transfers.

Leg	Stage	Val. sols per abs. sols		Number of solutions		Number of queries		Number of records		Time (seconds)	
		0-T	1-T	0-T	1-T	0-T	1-T	0-T	1-T	0-T	1-T
1	Turin → Madrid	0.5	0.667	2	1829	4	43	20	135	112.465	962.938
2	3 nights stay	1	1	6	6	1	1	20	20	62.384	62.384
3	Madrid → Toledo	0.5	0.333	12	797	4	43	22	92	75.030	1692.839
4	Toledo → Turin	0	0.333	0	432	4	43	0	67	76.236	3874.948

We have also tested a set of 38 problems with different configurations of MAPWEB. The problems include 15 trips within Spain, 15 within Europe, and 8 Intercontinental ones. Each problem has been tried with 0 and 1 transfers. The results are shown in Table 6. This experiment shows in practice the flexibility of MAPWEB when it is necessary to add new information sources. The configurations that have been used are as follows:

- *N0*: only one WebAgent specialized in retrieving information from a particular Airplane Company (Iberia Airlines[3]) was considered.
- *N1*: different WebAgents specialized in gathering information of the same kind (flight information) were used: WebAgent-Iberia, WebAgent-Avianca, WebAgent-Amadeus-Flights, WebAgent-4Airlines-Flights. The two last ones are meta-searchers.
- *N2*: only two WebAgents specialized in gathering information of the same type (train information) were used: WebAgent-Renfe, WebAgent-RailEurope.
- *N3*: integrates all the previous WebAgents, that is, agents for retrieving both flight and train information (*N3=N1+N2*).

Table 6. Summary of the results for 38 user problems, with 0 and 1 transfers (0-T and 1-T).

Config.	Number of solutions		Solved problems		Number of queries		Time (seconds)	
	0-T	1-T	0-T	1-T	0-T	1-T	0-T	1-T
N0	7.1	999.3	65.7%	74.3%	1	26.9	65.6	1485.7
	σ = 6.7	1480.5					σ = 10.2	1319.2
N1	10.9	1338.1	94.2%	97.1%	4	91.2	162.4	2243.6
	σ = 8.0	1725.8					σ = 200.5	1321.8
N2	3.7	3.7	25.7%	40.0%	2	51.4	70.1	1314.4
	σ = 7.9	7.9					σ = 23.0	1124.3
N3	12.5	1340.2	94.3%	97.1%	6	143.9	165.3	2666.6
	σ = 8.8	1724.4					σ = 199.6	1298.8

In Table 6, we observe the following:

- With respect to *N0*, as it could be expected, many more solutions are found when 1 transfer legs are allowed (999.3 vs. 7.1). It can also be observed that

[3] www.iberia.com/iberia_es/home.jsp

MAPWeb cannot find a solution for some problems, although the number of problems solved increases for the 1-T option (74.3% vs. 65.7%). However, the number of queries and the time required to fulfill them also increases quickly. It is also noticeable that standard deviations are rather large. This is because user problems can be very different; some of them can be solved quickly because there are few retrieved records, whereas other problems can have many possible solutions.

- *N1* enlarges *N0* by including more airplane companies. MAPWeb does not find many more solutions per problem, because most of the user problems are within Europe, where Iberia (the only agent in *N0*) offers many flights. However, many more problems are solved (94.2% vs. 65.7% with 0 transfers, and 97.1% vs. 65.7% for 1 transfer). Although the number of queries is multiplied by 4 in *N1*, the time required to fulfill them has been only doubled (162.4 vs. 65.6 for 0-T and 2243.6 vs. 1485.7 for 1-T). Time is doubled because even though the four WebAgents work in parallel, all the retrieved records must be analyzed by a single PlannerAgent.
- *N2* displays the results when only train travels are allowed. Only a few problems can be solved: 25.7% with 0-T and 40.0% with 1-T, and very few solutions per problem are found (3.7). This is clearly due to the smaller number of possibilities of fullfilling travels using only trains vs. using airplanes.
- *N3* integrates both airplane and train companies. Compared to *N1*, almost the same number of user problems are solved (94.3% vs. 94.2% and 97.1% vs. 97.1%), although some more solutions per problem are found (12.5 vs. 10.9 and 1340.2 vs. 1338.1).

5 Conclusions

The Web is a dynamic medium: more and more companies make their information available in the web everyday. Web information gathering systems need to be flexible to adapt to these rapid changes. In this paper we have described MAPWeb, a multiagent framework that combines classical planning techniques and Web information retrieval agents. MAPWeb decouples planning from information gathering, by splitting a planning problem into two parts: solving an abstract problem and validating and completing the abstract solutions by means of information gathering. Flexible information gathering is achieved by means of planning. In order to add a new information source to the system, only the planning domain has to be modified, besides adding the related Web agent.

In this paper MAPWeb has been applied to the e-tourism domain, but we believe it could be also used in other domains where planning can be separated from Web information gathering. For instance, currently many companies are thinking on moving to the Web and most organization process models will be implemented in such a way that they use information stored in the Web (either information internal to the organization or external). These processes can be automatically generated on-the-fly by planners, and they will need the information stored in the Web to decide on the steps to be performed. For

instance, one might define what information to publish (and how) in the WEB depending on the competence prices. This publishing process could be generated automatically by a planner.

In the future, several new skills will be developed for different agents in MAPWEB. These skills will try to improve the performance of the global system in two ways: by increasing the number and quality of solutions found by the agents, and by minimizing the time and computational resources used by MAPWEB to solve problems.

Acknowledgements

The research reported here was carried out as part of the research project funded by CICYT TAP-99-0535-C02.

References

1. Ambite, J.L., Knoblock, C.A.: Planning by rewriting: Efficiently generating high-quality plans. In proceedings of the Fourteenth National Conference on Artificial Intelligence (1997).
2. Camacho, D., Molina, J.M., Borrajo, D.: A Multiagent Approach for Electronic Travel Planning. Proceedings of the Second International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2000). AAAI. July (2000). Austin, TX (USA).
3. Camacho, D., Molina, J.M., Borrajo, D., Aler, R.: MAPWEB: Cooperation between Planning Agents and Web Agents. Information&Security: An International Journal. Special issue on Multi-agent Technologies. Volume 7 (2001).
4. Finin, T., Fritzson, R., Mackay, D., McEntire, R.: KQML as an Agent Communication Language. In Proceedings of the Third International Conference on Information and Knowledge Management (CIKM94), pages 456–463. New York: Association of Computing Machinery (1994).
5. Hüllen, J., Bergmann, R., Weberskirch, F: WebPlan - Dynamic planning for domain-specific search in the Internet. In J. Köhler (Hrsg.) 13. Workshop "Planen und Konfigurieren". (PuK-99) (1999).
6. Lambrecht, E., Kambhampati, S.: Planning for Information Gathering: A tutorial Survey. ASU CSE Techincal Report 96-017. May (1997).
7. Knoblock, C.A., Minton, S., Ambite, J.L., Ashish, N.: Modeling Web Sources for Information Integration. Proceedings of the Fifteenth National Conference on Artificial Intelligence, 1998.
8. Knoblock, C.A., Minton, S., Ambite, J.L., Muslea, M., Oh, J., Frank, M.: Mixed-Initiative, Multi-source Information Assistants. The Tenth International World Wide Web Conference (WWW10). ACM Press. May 1-5. (2001).
9. Veloso, M., Carbonell, J., Perez, A. Borrajo, D., Fink, E., Blythe, J.: Integrating planning and learning: The Prodigy architecture. Journal of Experimental and Theoretical AI. Volume 7 (1995) 81–120.